

CS 61C: Great Ideas in Computer Architecture (Machine Structures)

Set-Associative Caches

Instructors:
Randy H. Katz
David A. Patterson

<http://inst.eecs.Berkeley.edu/~cs61c/fa10>

11/10/10 Fall 2010 – Lecture #31 1

Agenda

- Cache Memory Recap
- Administrivia
- Technology Break
- Set Associative Caches

11/10/10 Fall 2010 – Lecture #31 2

Agenda

- Cache Memory Recap
- Administrivia
- Technology Break
- Set Associative Caches

11/10/10 Fall 2010 – Lecture #31 3

Recap: Components of a Computer

11/10/10 Fall 2010 – Lecture #31 4

Recap: Typical Memory Hierarchy

- Take advantage of the **principle of locality** to present the user with as much memory as is available in the *cheapest* technology at the speed offered by the *fastest* technology

| | | | | | |
|-------------------------|---------|-------|------|-------|----------|
| | 1/2's | 1's | 10's | 100's | 10,000's |
| Speed (%cycles): | 1/2's | 1's | 10's | 100's | 10,000's |
| Size (bytes): | 100's | 10K's | M's | G's | T's |
| Cost: | highest | | | | lowest |

11/10/10 Fall 2010 – Lecture #31 5

Recap: Cache Performance and Average Memory Access Time (AMAT)

- CPU time = IC × CPI × CC
- = IC × (CPI_{ideal} + Memory-stall cycles) × CC
- CPI_{stall}

Memory-stall cycles = Read-stall cycles + Write-stall cycles

Read-stall cycles = reads/program × read miss rate × read miss penalty

Write-stall cycles = (writes/program × write miss rate × write miss penalty) + write buffer stalls

- AMAT is the average time to access memory considering both hits and misses
- AMAT = Time for a hit + Miss rate × Miss penalty

11/10/10 Fall 2010 – Lecture #31 6

Improving Cache Performance

- Reduce the time to hit in the cache
 - E.g., Smaller cache, direct mapped cache, smaller blocks, special tricks for handling for writes
- Reduce the miss rate
 - E.g., Bigger cache, larger blocks
 - More flexible placement (increase associativity)
- Reduce the miss penalty
 - E.g., Smaller blocks or critical word first in large blocks, special tricks for handling for writes, faster/higher bandwidth memories
 - Use multiple cache levels

11/10/10 Fall 2010 – Lecture #31 7

Sources of Cache Misses: The 3Cs

- **Compulsory** (cold start or process migration, 1st reference):
 - First access to block impossible to avoid; small effect for long running programs
 - Solution: increase block size (increases miss penalty; very large blocks could increase miss rate)
- **Capacity**:
 - Cache cannot contain all blocks accessed by the program
 - Solution: increase cache size (may increase access time)
- **Conflict** (collision):
 - Multiple memory locations mapped to the same cache location
 - Solution 1: increase cache size
 - Solution 2: increase associativity (may increase access time)

11/10/10 Fall 2010 – Lecture #31 8

Reducing Cache Misses

- Allow more flexible block placement
- **Direct mapped** \$: memory block maps to exactly one cache block
- **Fully associative** \$: allow a memory block to be mapped to any cache block
- Compromise: divide \$ into sets, each of which consists of n “ways” (**n-way set associative**) to place memory block
 - Memory block maps to unique set determined by index field and is placed in any of the n-ways of that set
 - Calculation: (block address) modulo (# sets in the cache)

11/10/10 Fall 2010 – Lecture #31 9

Alternative Block Placement Schemes

Direct mapped

Set associative

Fully associative

- DM placement: mem block 12 in 8 block cache: only one cache block where mem block 12 can be found—(12 modulo 8) = 4
- SA placement: four sets x 2-ways (8 cache blocks), memory block 12 in set (12 mod 4) = 0; either element of the set
- FA placement: mem block 12 can appear in any cache blocks

11/10/10 Fall 2010 – Lecture #31 10

Agenda

- Cache Memory Recap
- Administrivia
- Technology Break
- Set Associative Caches

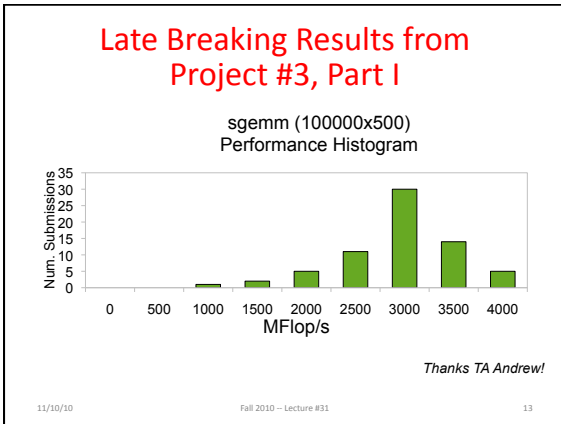
11/10/10 Fall 2010 – Lecture #31 11

Late Breaking Results from Project #3, Part I

sgemm (10000x500)
3 levels blocking, loop unrolling

Thanks TA Andrew!

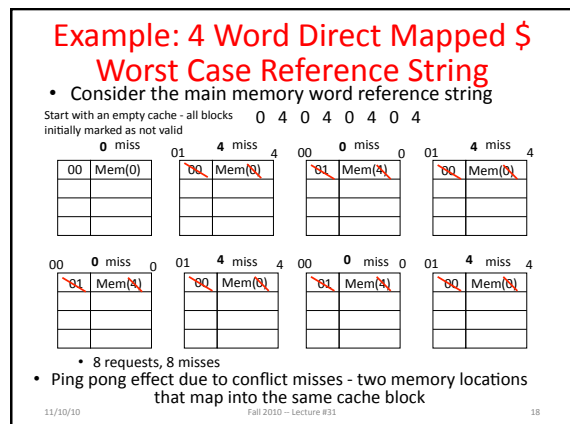
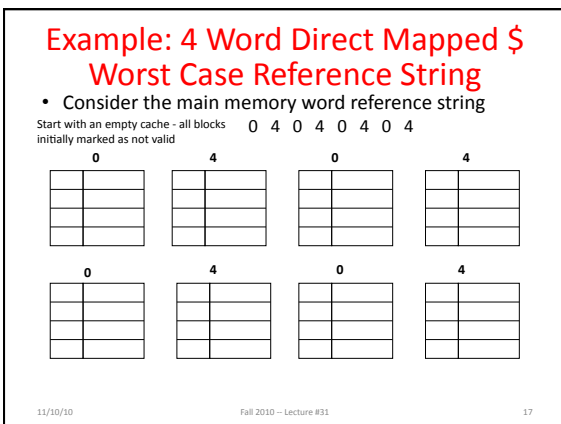
11/10/10 Fall 2010 – Lecture #31 12



- ### Administrivia
- Posted Optional Extra Logicsim "At Home" Lab this week
 - Project 3: TLP+DLP+Cache Opt (Due 11/13)
 - Project 4: Single Cycle Processor in Logicsim (by tonight)
 - Due Part 2 due Saturday 11/27
 - Face-to-Face in lab 12/2
 - EC: Fastest Project 3 (due 11/29)
 - Final Review: Mon Dec 6, 3 hrs, afternoon (TBD)
 - Final: Mon Dec 13 8AM-11AM (TBD)
 - Like midterm: T/F, M/C, short answers
 - Whole Course: readings, lectures, projects, labs, hw
 - Emphasize 2nd half of 61C + midterm mistakes
- 11/10/10 Fall 2010 -- Lecture #30 14

- ### Agenda
- Cache Memory Recap
 - Administrivia
 - Technology Break
 - Set-Associative Caches
- 11/10/10 Fall 2010 -- Lecture #31 15

- ### Agenda
- Cache Memory Recap
 - Administrivia
 - Technology Break
 - Set Associative Caches
- 11/10/10 Fall 2010 -- Lecture #31 16



Example: 2-Way Set Associative \$ (4 words = 2 sets x 2 ways per set)

Cache

| Way | Set | V | Tag | Data |
|-----|-----|---|-----|------|
| 0 | 0 | | | |
| 0 | 1 | | | |
| 1 | 0 | | | |
| 1 | 1 | | | |

Q: Is it there?

Compare *all* the cache tags in the set to the high order 3 memory address bits to tell if the memory block is in the cache

Main Memory

| |
|--------|
| 000Dxx |
| 000Lxx |
| 000Dxx |
| 001Lxx |
| 010Dxx |
| 010Lxx |
| 011Dxx |
| 011Lxx |
| 100Dxx |
| 100Lxx |
| 101Dxx |
| 101Lxx |
| 110Dxx |
| 110Lxx |
| 111Dxx |
| 111Lxx |

One word blocks
Two low order bits define the byte in the word (32b words)

Q: How do we find it?

Use next 1 low order memory address bit to determine which cache set (i.e., modulo the number of sets in the cache)

11/10/10 Fall 2010 - Lecture #31 19

Example: 4 Word 2-Way SA \$ Same Reference String

- Consider the main memory word reference string

Start with an empty cache - all blocks initially marked as not valid

0 4 0 4 0 4 0 4

| | | | |
|---|---|---|---|
| 0 | 4 | 0 | 4 |
| | | | |
| | | | |
| | | | |

11/10/10 Fall 2010 - Lecture #31 20

Example: 4 Word 2-Way SA \$ Same Reference String

- Consider the main memory word reference string

Start with an empty cache - all blocks initially marked as not valid

0 4 0 4 0 4 0 4

| | | | | | | | |
|-----|--------|-----|--------|-----|--------|-----|--------|
| 000 | Mem(0) | 000 | Mem(0) | 000 | Mem(0) | 000 | Mem(0) |
| | | 010 | Mem(4) | 010 | Mem(4) | 010 | Mem(4) |

- 8 requests, 2 misses
- Solves the ping pong effect in a direct mapped cache due to conflict misses since now two memory locations that map into the same cache set can co-exist!

11/10/10 Fall 2010 - Lecture #31 21

Example: Eight Block Cache with Different Organizations

One-way set associative (direct mapped)

| Block | Tag | Data |
|-------|-----|------|
| 0 | | |
| 1 | | |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| 6 | | |
| 7 | | |

Two-way set associative

| Set | Tag | Data | Tag | Data |
|-----|-----|------|-----|------|
| 0 | | | | |
| 1 | | | | |
| 2 | | | | |
| 3 | | | | |

Four-way set associative

| Set | Tag | Data | Tag | Data | Tag | Data | Tag | Data |
|-----|-----|------|-----|------|-----|------|-----|------|
| 0 | | | | | | | | |
| 1 | | | | | | | | |

Eight-way set associative (fully associative)

| Tag | Data | Tag | Data | Tag | Data | Tag | Data | Tag | Data | Tag | Data | Tag | Data |
|-----|------|-----|------|-----|------|-----|------|-----|------|-----|------|-----|------|
| | | | | | | | | | | | | | |

Total size of \$ in blocks is equal to number of sets x associativity. For fixed \$ size, increasing associativity decreases number of sets while increasing number of elements per set. With eight blocks, an 8-way set-associative \$ is same as a fully associative \$.

11/10/10 Fall 2010 - Lecture #31 22

Four-Way Set Associative Cache

- $2^8 = 256$ sets each with four ways (each with one block)

11/10/10 Fall 2010 - Lecture #31 23

Range of Set Associative Caches

- For a fixed size cache, each increase by a factor of two in associativity doubles the number of blocks per set (i.e., the number of ways) and halves the number of sets - decreases the size of the index by 1 bit and increases the size of the tag by 1 bit

| | | | |
|-----|-------|--------------|-------------|
| Tag | Index | Block offset | Byte offset |
|-----|-------|--------------|-------------|

11/10/10 Fall 2010 - Lecture #31 24

Range of Set Associative Caches

- For a fixed size cache, each increase by a factor of two in associativity doubles the number of blocks per set (i.e., the number or ways) and halves the number of sets – decreases the size of the index by 1 bit and increases the size of the tag by 1 bit

Used for tag compare Selects the set Selects the word in the block

Tag Index Block offset Byte offset

Decreasing associativity ← → Increasing associativity

Direct mapped (only one way)
Smaller tags, only a single comparator

Fully associative (only one set)
Tag is all the bits except block and byte offset

11/10/10 Fall 2010 – Lecture #31 25

Costs of Set Associative Caches

- When miss occurs, which way's block selected for replacement?
 - Least Recently Used (LRU):** one that has been unused the longest
 - Must track when each way's block was used relative to other blocks in the set
 - For 2-way SA \$, one bit per set → set to 1 when a block is referenced; reset the other way's bit (i.e., "last used")
- N-way set associative cache costs
 - N comparators (delay and area)
 - MUX delay (set selection) before data is available
 - Data available after set selection (and Hit/Miss decision).
 - DM \$: block is available before the Hit/Miss decision
 - Not possible to just assume a hit and continue and recover later if it was a miss

11/10/10 Fall 2010 – Lecture #31 26

Cache Block Replacement Policies

- Random Replacement
 - Hardware randomly selects a cache item and throw it out
- Least Recently Used
 - Hardware keeps track of access history
 - Replace the entry that has not been used for the longest time
 - For 2-way set-associative cache, need one bit for LRU replacement
- Example of a Simple "Pseudo" LRU Implementation
 - Assume 64 Fully Associative entries
 - Hardware replacement pointer points to one cache entry
 - Whenever access is made to the entry the pointer points to:
 - Move the pointer to the next entry
 - Otherwise: do not move the pointer

| |
|----------|
| Entry 0 |
| Entry 1 |
| ⋮ |
| Entry 63 |

Replacement Pointer

11/10/10 Fall 2010 – Lecture #31 27

Benefits of Set Associative Caches

- Choice of DM \$ or SA \$ depends on the cost of a miss versus the cost of implementation

Miss rate

One-way Two-way Four-way Eight-way

1 KB, 2 KB, 4 KB, 8 KB, 16 KB, 32 KB, 64 KB, 128 KB

- Largest gains are in going from direct mapped to 2-way (20%+ reduction in miss rate)

11/10/10 Fall 2010 – Lecture #31 28

3Cs Revisted

Miss rate per type

Cache size (KB)

One-way, Two-way, Four-way

- Three sources of misses (SPEC2000 integer and floating-point benchmarks)
 - Compulsory misses 0.006%; not visible
 - Capacity misses, function of cache size
 - Conflict portion depends on associativity and cache size

11/10/10 Fall 2010 – Lecture #31 29

Two Machines' Cache Parameters

| | Intel Nehalem | AMD Barcelona |
|------------------------------|---|---|
| L1 cache organization & size | Split IS and DS; 32KB for each per core; 64B blocks | Split IS and DS; 64KB for each per core; 64B blocks |
| L1 associativity | 4-way (I), 8-way (D) set assoc.; ~LRU replacement | 2-way set assoc.; LRU replacement |
| L1 write policy | write-back, write-allocate | write-back, write-allocate |
| L2 cache organization & size | Unified; 256MB (0.25MB) per core; 64B blocks | Unified; 512KB (0.5MB) per core; 64B blocks |
| L2 associativity | 8-way set assoc.; ~LRU | 16-way set assoc.; ~LRU |
| L2 write policy | write-back | write-back |
| L2 write policy | write-back, write-allocate | write-back, write-allocate |
| L3 cache organization & size | Unified; 8192KB (8MB) shared by cores; 64B blocks | Unified; 2048KB (2MB) shared by cores; 64B blocks |
| L3 associativity | 16-way set assoc. | 32-way set assoc.; evict block shared by fewest cores |
| L3 write policy | write-back, write-allocate | write-back; write-allocate |

11/10/10 Fall 2010 – Lecture #31 30

Summary

- Name of the Game: Reduce Cache Misses
 - Two different memory blocks mapping to same cache block could knock each other out as program bounces from one memory location to the next
- One way to do it: set-associativity
 - Memory block maps into more than one cache block
 - N-way: n possible places in the cache to hold a given memory block
 - N-way Cache of 2^{N+M} blocks: 2^N ways x 2^M sets